5 VALUES API Usage Examples

5.1 Overview

This chapter gives a single programming example with code fragments in a combination of ANSI C and pseudo-code that describe the usage of the individual entry points. The programming example consists of code fragments and explanatory text and describes the usage of all VALUES call interface entry points. The code fragments include calls to the VALUES API, parameter specification, memory allocation, and examples of application callbacks.

The chapter is split into sections outlining the path from initiation of a session, logging into Deutsche Börse Back End applications, application requests/responses, subscribing and receiving of subscription data, un-subscribing, logging out Deutsche Börse Back End applications, and termination of a session.

The last two sections in the chapter show how to set up an application’s Dispatch loop. One example assumes a Sun platform, the other example is based upon Galaxy event handling.

5.2 Initiating a VALUES Session

The VCI_Connect entry point is used to connect the application to VALUES. After a successful VCI_Connect, the application can send subscription application requests, or perform login into the Back End systems.

```
// code example VCI_Connect
#include “Values.h”
#include “ehs_konst.h”
#include “requests.h”

// global definitions: the stack persistency is not enough for these variables as they must be used across multiple
// application functions
int             myConnectionID;     // myConnectionId identifies the session and
                                    // is mandatory input for all future interface calls
char            prodMode;           // application can use prodMode to inform
                                    // the user about the production mode

void connect_function()
{
        // local variables
        int             myVMQ;

        // define variables in VCI-format
        ReqCntrlT       reqControl;
        CnctReqDataT    *reqData;
        CnctRespDataT           respData;
        StatusDataT     statusDataGlobal;
        AppCntxtDataT   *callbackContextData;

        // fill the field ‘VCIver’ of reqControl
        // VCI_VERSION is a VALUES constant
        strcpy(reqControl.VCIver, VCI_VERSION);         // set VALUES version for compatibility checking

        // allocate memory for reqData (this memory has to be deallocated after disconnecting)
        reqData = (CnctReqDataT *) malloc(sizeof(CnctReqDataT));

        // fill the fields of the reqData-structure
        // the userId and password should come from a user or a file and not be
        // hardcoded
        strcpy(reqData->userID, “johnson“);     // FE operating system userID for VALUES or
                                                // DB-application
        strcpy(reqData->password, “pencil“);    // FE operating system password for userID

        // allocate memory for callbackContextData (this memory has to be deallocated after disconnecting)
        callbackContextData = (AppCntxtDataT *) malloc(sizeof(AppCntxtDataT));
```

```
// fill the fields of the callbackContextData-structure
// This example uses no context data
callbackContextData->custBlockSize = 0;

// call VCI_Connect
VCI_Connect    (        &reqControl,
                        reqData,
                        connectCallback,
                        callbackContextData,
                        &respData,
                        &statusDataGlobal
                );

// Assess whether or not the connection request succeeded
switch (statusDataGlobal.complSeverity)
{
        case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                // return-data received
                if (respData)
                {
                        // save response-data for use in later VALUES processing
                        // the connection ID is stored
                        myConnectionID = respData.connectionID;

                        // retrieval of the VMQ identifier
                        // myVMQ may subsequently be used to register an event and
                        // callback for the application's Dispatch event loop
                        myVMQ = atoi( respData.VMQname );

                        // storage of the Production Mode
                        prodMode = respData.prodMode;
                }
                break;
        case VCI_FATAL:         // VCI_FATAL = Fatal error has
                                // occured. The application should perform a shutdown.
                fatality_handling(statusDataGlobal);
                break;
        case VCI_ERROR:         // VCI_ERROR = An error has
                                // been detected. The application has to perform
                                // error-handling processing depending on which
                                // completion code has  been returned
                error_handling(statusDataGlobal);
                break;
        case VCI_WARNING:   // VCI_WARNING = A minor
                                // error occured. The application may have to
                                // perform error-handling processing depending on
                                // which completion code has been returned.
                warning_handling(statusDataGlobal);
                break;
        }       // end switch
}       // end connect_function()
```

```
// callback used to handle asynchronous events related to the application's connection to VALUES
void connectCallback(
        ReqCntrlT               *reqControl,
        CallBkAppDataT          *appData,
        AppCntxtDataT                   *callBackCntxtData,
        StatusDataT             *statusDataGlobal,
        )
{
        controlMessageT *respCntrl = (controlMessageT*) appData->appRespData;
        switch(respCntrl->controlMsgCode)
        {
                case ELB_IBISR_STARTUP:
                    //exception handling has to be performed here...
                case ELB_IBISR_SHUTDOWN:
                    //exception handling has to be performed here...
                case ELB_XETRA_STARTUP:
                    //exception handling has to be performed here...
                case ELB_XETRA_SHUTDOWN:
                    //exception handling has to be performed here...
        }
        // end exceptionCallback()
}
// End code example VCI_Connect
```

5.3 Logging on to Deutsche Börse Back End Applications

The VCI_Login entry point is used to login into Deutsche Börse applications. Before sending any dialog or inquiry application request, the user must get the authorization from the appropriate Deutsche Börse application (IBIS-R, Xetra®).

```
// code example VCI_Login
#include “Values.h”
#include “ehs_konst.h”

// global definitions: the stack persistency is not enough for these variables as they must be used across multiple
// application functions
int     myConnectionID;     // myConnectionId identifies the session and
                            // is mandatory input for all future interface calls

void login_function()
{

        // my own variables

        // define variables in VCI-format
        ReqCntrlT           reqControl;

        IbisRLoginDataT         *reqData;
        LoginRespDataT          respData;
        StatusDataT         statusDataGlobal;
        AppCntxtDataT       *callbackContextData;

        // fill the mandatory fields of VCIver
        reqControl.dbApplID = IBISr_id;         // logon is to be done for IBIS-R
        strcpy(reqControl.VCIver, VCI_VERSION);     // set VALUES version for compatibility
                                                    // checking

        // at this point, the connection ID obtained as a returned parameter
        // from VCI_Connect is used to identify the session
        reqControl.connectionID = myConnectionID;

        // allocate memory for reqData (this memory has to be deallocated after the logout)
        reqData = (IbisRLoginDataT *) malloc(sizeof(IbisRLoginDataT));

        // fill the fields of the reqData-structure
        // the userID and password should come from a file or a user, but not be hardcoded
        strcpy(reqData->userID, “johnson“);             // IBIS-R userID
        strcpy(reqData->password, “napoleon“);                  // IBIS-R password

        // allocate memory for callbackContectData (this memory has to be deallocated after the logout)
        callbackContextData = (AppCntxtDataT *) malloc(sizeof(AppCntxtDataT));
```

```
        // fill the fields of the callbackContextData-structure
        // This example uses no context data
        callbackContextData->custBlockSize = 0;

        // call VCI_Login
        VCI_Login(      &reqControl,
                        reqData,
                        loginCallback,
                        callbackContextData,
                        &respData,
                        &statusDataGlobal
                );

        // Assess whether or not the login request succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:   // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:         // VCI_FATAL = Fatal error has
                                        // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:         // VCI_ERROR = An error has
                                        // been detected. The application has to perform
                                        // error-handling processing depending on which
                                        // completion code has  been returned
                        error_handling(statusDataGlobal);
                        break;
                case VCI_WARNING:   // VCI_WARNING = A minor
                                        // error occured. The application may have to
                                        // perform error-handling processing depending on
                                        // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        }       // end switch
}       // end login_function()

// callback used to handle asynchronous events related to the application's login to VALUES
void loginCallback(
        ReqCntrlT               *reqControl,
        CallBkAppDataT                  *appData,
        AppCntxtDataT           *callBackCntxtData,
        StatusDataT             *statusDataGlobal,
        )
{
        // login callback handles exception cases
}

// End code example VCI_Login
```

5.4 Submitting Application Requests

The VCI_Submit call interface entry point is used by end user application to send processing requests to Deutsche Börse applications. The user has to specify a request code, request data, and an application callback.

```
// code example VCI_Submit (request)
#include “Values.h”
#include “ehs_konst.h”
#include “app_rid.h”
#include “requests.h”

// global definitions: the stack persistency is not enough for these variables as they must be used across multiple
// application functions
int                 myConnectionID;          // myConnectionId identifies the session and
                                             // is mandatory input for all future interface calls
char                myContextData[100];      // the connectCallback receives a reference to
                                             // myContextData in callbackCntxtData
                                             // from VALUES and needs to access this data.
                                             // The data has to be valid until the response
                                             // is handled
s_enter_offer_ibis_r my_application_request_structure;   // the responseCallback
                                             // receives a reference appReq =
                                             // &my_application_request_structure in order to
                                             // identify the request that triggered the response
                                             // therefore, my_application_request_structure
                                             // must be stored, for example, by declaring it as
                                             // a global variable . The data has to be valid until
                                             // the response is handled.


void submit_function()
{
        // local variables

        // define variables in VCI-format
        ReqCntrlT           reqControl;
        SubmitReqDataT              *reqData;
        StatusDataT         statusDataGlobal;
        AppCntxtDataT       *callbackCntxtData;

        // fill the fields of reqControl
        reqControl.dbApplID = IBISr_id;                  // a request to IBIS-R is submitted
        reqControl.reqID = ENTER_OFFER_IBISR_RID;        // set the type of the request
        strcpy(reqControl.VCIver, VCI_VERSION);          // set VALUES version for
                                                         // compatibility checking
```

```
// at this point, the connection ID obtained as a returned parameter
// from VCI_Connect is used to identify the session
reqControl.connectionID = myConnectionID;

// the filling of the application request structure should be done at this point
// in general these fields are not hardcoded, but entered by a user
strcpy(my_application_request_structure.wkn, “0123456”);        // set instrument number
strcpy(my_application_request_structure.buySellInd, “B”);                // set indicator to Buy
strcpy(my_application_request_structure.prc, “100”);            // set price
strcpy(my_application_request_structure.qty, “13”);             // set quantity
 // similarly, the other fields of the application request structure are filled at this point

// allocate memory for reqData (this memory has to be deallocated in the responseCallback)
reqData = (SubmitReqDataT *) malloc(sizeof(SubmitReqDataT));

// fill reqData
reqData ->appReq = &my_application_request_structure;          //pass reference to VALUES
                                                    // this reference is passed back to the
                                                    // application in the application callback and
                                                    // may be used to identify the request that
                                                    // caused the response callback
reqData ->appReqBlockSize = sizeof(my_application_request_structure);
                                                    // set size of context data, so response
                                                    // callback knows size of information
                                                    // stored in appReqData

// fill custom data
strcpy(myContextData, “this is a context data”); // store context data to be accessed in
                                                    // the response callback

// allocate memory for callbackContextData (this memory has to be deallocated in the responseCallback)
callbackContextData = (AppCntxtDataT *) malloc(sizeof(AppCntxtDataT));

// customize callbackContextData
callBackContextData->custData = myContextData;       // Pass reference to context data to
                                                    // VALUES. This reference is passed back to the
                                                    // application in the application callback.
CallBackContextData->custBlockSize = sizeof(myContextData);

// call VCI_Submit
VCI_Submit(     &reqControl,
                reqData,
                responseCallback,
                callbackContextData,
                &statusDataGlobal
          );
```

```
        // Assess whether or not the call succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:      // VCI_FATAL = Fatal error has
                                     // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:      // VCI_ERROR = An error has
                                     // been detected. The application has to perform
                                     // error-handling processing depending on which
                                     // completion code has  been returned
                        error_handling(statusDataGlobal);
                        break;
                case VCI_WARNING:    // VCI_WARNING = A minor
                                     // error occured. The application may have to
                                     // perform error-handling processing depending on
                                     // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        }       // end switch
}       // end submit_function()

// an example of the responseCallback is given in section 5.6

// End code example VCI_Submit
```

5.5 Application Dispatch upon Event Notification

The VCI_dispatch entry point is used by the application to service application responses, broadcasts and exceptions. VCI_Dispatch will read response data from the VMQ and pass it on the appropriate application response callback.

```
// code example VCI_Dispatch
#include “Values.h”
#include “ehs_konst.h”

// global definitions: the stack persistency is not enough for these variables as they must be used across multiple
// application functions
int                     myConnectionID;         // myConnectionId identifies the session and
                                                // is mandatory input for all future interface calls
void dispatch_function()
{
        // define variables in VCI-format
        ReqCntrlT               reqControl;
        StatusDataT             statusDataGlobal;

        // fill the field ‘VCIver’ of reqControl
        strcpy(reqControl.VCIver, VCI_VERSION);       // set VALUES version for compatibility
                                                      // checking

        // at this point, the connection ID obtained as a returned parameter
        // from VCI_Connect is used to identify the session
        reqControl.connectionID = myConnectionID;

        // call VCI_Dispatch
        VCI_Dispatch(  &reqControl,
                       &statusDataGlobal
                       );
```

```
        // Assess whether or not the call succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:      // VCI_FATAL = Fatal error has
                                     // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:      // VCI_ERROR = An error has
                                     // been detected. The application has to perform
                                     // error-handling processing depending on which
                                     // completion code has  been returned
                        error_handling(statusDataGlobal);
                        break;

                case VCI_WARNING:    // VCI_WARNING = A minor
                                     // error occured. The application may have to
                                     // perform error-handling processing depending on
                                     // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        }       // end switch

}       // end dispatch_function()
// End code example VCI_Dispatch
```

5.6 Receiving Application Responses

These callbacks are implemented in the application to be invoked by VALUES to perform asynchronous processing in response to application requests.

```
// code example for handling the callback of an application response
#include “Values.h”
#include “ehs_konst.h”
#include “app_rid.h”
#include “ehs_strukt.h”

void responseCallback( ReqCntrlT          *reqControl,
                       CallBkAppDataT          *appData,
                       AppCntxtDataT      *appCntxtData,
                       StatusDataT        *statusDataGlobal,
                       )
{
        int                    responseSize;
        s_enter_offer_ibis_r   *my_application_request_structure;
        E_EOR_R                *responseData;

        // The reqControl structure contains the reqID of the request that generates
        // this callback. The reqID may be retrieved at this point to determine further processing.

        //The structure appReqData of the appData parameter
        // contains the request that was sent by the application with the
        // VCI_Submit call
        // This information can be retrieved as follows:
        my_application_request_structure = (s_enter_offer_ibis_r*) appData->appReqData;

        //The structure appData contains the application response data and the size of response data:
        responseSize = appData.appRespBlockSize;
        responseData = (E_EOR_R*) appData->appRespData;

        //The structure appCntxtData contains the application context data, which can
        // be retrieved  similar to the appReqData, or the appRespData
```

```
        // Assess whether or not the call succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:      // VCI_FATAL = Fatal error has
                                     // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:      // VCI_ERROR = An error has
                                     // been detected. The application has to perform
                                     // error-handling processing depending on which
                                     // completion code has  been returned
                        error_handling(statusDataGlobal);
                        break;
                case VCI_WARNING:    // VCI_WARNING = A minor
                                     // error occured. The application may have to
                                     // perform error-handling processing depending on
                                     // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        } // end switch
        // deallocation of appCntxtData and appReqData
        free(appCntxtData);
        free(appData->appReqData);

}       // end responseCallback()
```

5.7 Subscribing to a Data Stream

The VCI_Subscribe call is used by the application to subscribe to data streams. Before subscribing, an application must have established a session. The user must specify the desired data stream, and an application callback.

```
// code example VCI_Subscribe
#include “Values.h”
#include “ehs_konst.h”
#include “app_rid.h”
#include “ehs_strukt.h”

// global definitions: the stack persistency is not enough for these variables as they must be used across multiple
// application functions
int             myConnectionID;     // myConnectionId identifies the session and
                                    // is mandatory input for all future interface calls
int             mySubsId;           // VCI_Subscribe returns a subscription ID
                                    // which is needed to un-subscribe at a later
                                    // stage in the application

void subscribe_function()
{
        // define variables in VCI-format
        ReqCntrlT           reqControl;
        SubsReqDataT        *reqData;
        AppCntxtDataT       *callbackContextData;
        SubsRespDataT               respData;
        StatusDataT         statusDataGlobal;

        // fill the reqControl parameters
        strcpy(reqControl.VCIver, VCI_VERSION);         // set VALUES version for compatibility
                                                        // checking

        // at this point, the connection ID obtained as a returned parameter
        // from VCI_Connect is used to identify the session
        reqControl.connectionID = myConnectionID;

        // allocate memory for reqData (this memory has to be deallocated in after calling VCIUnsubscribe)
        reqData = (SubsReqDataT *) malloc(sizeof(SubsReqDataT));

        // fill the fields of the reqData-structure
        // these fields should not be hardcoded, but entered by a user, or retrieved from a file
        reqData->subsSubject.product = IbisRMarket;
        strcpy(reqData->subsSubject.type.ibisR.trdrId, “567890“);
        strcpy(reqData->subsSubject.type.ibisR.mbrId, “1234“);
        strcpy(reqData->subsSubject.type.ibisR.wkn, “0123456”);
        reqData->subsSubject.type.ibisIbisR.qualifier = wildcard;

        // allocate memory for callbackContextData (this memory has to be deallocated after calling VCIUnsubscribe)
        callbackContextData = (AppCntxtDataT *) malloc(sizeof(AppCntxtDataT));
```

```
        // this example does not use callbackContextData
        callBackContextData->custBlockSize = 0;

        // call VCI_Subscribe
        VCI_Subscribe (&reqControl,
                       reqData,
                       broadcastCallback,
                       callbackContextData,
                       &respData,
                       &statusDataGlobal
                       );

        // store subscription id for later unsubscribing
        mySubsId = respData.subsID;

        // Assess whether or not the call succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:      // VCI_FATAL = Fatal error has
                                     // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:      // VCI_ERROR = An error has
                                     // been detected. The application has to perform
                                     // error-handling processing depending on which
                                     // completion code has  been returned
                        error_handling(statusDataGlobal);
                        break;
                case VCI_WARNING:    // VCI_WARNING = A minor
                                     // error occured. The application may have to
                                     // perform error-handling processing depending on
                                     // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        }       // end switch
}       // end subscribe_function()

// An example of the broadcastCallback is included in section 5.8

// End code example VCI_Subscribe
```

5.8 Receiving Subscription Data

These callbacks are implemented in the application to be invoked by VALUES to perform asynchronous processing in response to subscription requests.

```
// code example for handling the callback of an subscription response
#include “Values.h”
#include “ehs_konst.h”
#include “app_rid.h”
#include “ehs_strukt.h”

void broadcastCallback(
                ReqCntrlT               *reqData,
                CallBkAppDataT                  *appData,
                AppCntxtDataT           *appCntxtData,
                StatusDataT             *statusDataGlobal,
                )
{
        int             responseSize;
        instrIbisr      *responseData;

        // The structure “subject” of the appData parameter
        // contains the subject of this broadcast message
        // and the subscription ID.
        // This subject is identical to the subject used to subscribe to this stream,
        // except that wildcards are replaced by actual values.
        // Code to retrieve the subject and subscription ID can be written here:

        // The structure appData contains the application response data and the size of response data.
        responseSize = appData.appReqBlockSize;
        responseData = (instrIbisr*)appData.appRespData;

        // Context data may be read at this point
        // This example contains no context data
```

```
        // Assess whether or not the call succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:      // VCI_FATAL = Fatal error has
                                     // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:      // VCI_ERROR = An error has
                                     // been detected. The application has to perform
                                     // error-handling processing depending on which
                                     // completion code has  been returned
                        error_handling(statusDataGlobal);
                        break;
                case VCI_WARNING:    // VCI_WARNING = A minor
                                     // error occured. The application may have to
                                     // perform error-handling processing depending on
                                     // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        }       // end switch

}       // end broadcastCallback()

// end code example broadcastCallback
```

5.9 Unsubscribing to a Data Stream

The VCI_Unsubscribe entry point is used by the application to end the subscription to a data stream.

```
// code example VCI_Unsubscribe
#include “Values.h”
#include “ehs_konst.h”
#include “app_rid.h”

// global definitions: the stack persistency is not enough for these variables as they must be used across multiple
// application functions
int             myConnectionID;     // myConnectionId identifies the session and
                                    // is mandatory input for all future interface calls
int             mySubsId;           // the subscription ID is returned by
                                    // VCI_Subscribe and stored to un-subscribe

void unsubscribe_function()
{
        // define variables in VCI-format
        ReqCntrlT           reqControl;
        UnsubsReqDataT      reqData;
        UnsubsRespDataT     respData;
        StatusDataT         statusDataGlobal;

        // fill the reqControl parameters
        strcpy(reqControl.VCIver, VCI_VERSION);     // set VALUES version for compatibility checking

        // at this point, the connection ID obtained as a returned parameter
        // from VCI_Connect is used to identify the session
        reqControl.connectionID = myConnectionID;

        // the stored subscription ID is needed to unsubscribe.
        reqData.subsID = mySubsId;

        // call VCI_Unsubscribe
        VCI_Unsubscribe(&reqControl,
                        &reqData,
                        &respData,
                        &statusDataGlobal
                        );
```

```
        // Assess whether or not the call succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:      // VCI_FATAL = Fatal error has
                                     // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:      // VCI_ERROR = An error has
                                     // been detected. The application has to perform
                                     // error-handling processing depending on which
                                     // completion code has  been returned
                        error_handling(statusDataGlobal);
                        break;
                case VCI_WARNING:    // VCI_WARNING = A minor
                                     // error occured. The application may have to
                                     // perform error-handling processing depending on
                                     // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        }       // end switch

}       // end unsubscribe_function()

// End code example VCI_Unsubscribe
```

5.10 Logging off from Deutsche Börse Back End Applications

The VCI_logout entry point is used by the application to logout from Deutsche Börse applications.

```
// code example VCI_Logout
#include “Values.h”
#include “ehs_konst.h”

// global definitions: the stack persistency is not enough for these variables as they must be used across multiple
// application functions
int     myConnectionID;     // myConnectionId identifies the session and
                            // is mandatory input for all future interface calls

void logout_function()
{
        // define variables in VCI-format
        ReqCntrlT           reqControl;
        LogoutReqDataT          reqData;
        LogoutRespDataT     respData;
        StatusDataT         statusDataGlobal;

        // fill the fields of reqControl
        reqControl.dbApplId = IBISr_id;         // specify a logout from IBIS-R
        strcpy(reqControl.VCIver, VCI_VERSION);     // set VALUES version for compatibility checking

        // at this point, the connection ID obtained as a returned parameter
        // from VCI_Connect is used to identify the session
        reqControl.connectionID = myConnectionID;

        // call VCI_Logout
        VCI_Logout(     &reqControl,
                        &reqData,
                        &respData,
                        &statusDataGlobal
                        );
```

```
        // Assess whether or not the call succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:      // VCI_FATAL = Fatal error has
                                     // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:      // VCI_ERROR = An error has
                                     // been detected. The application has to perform
                                     // error-handling processing depending on which
                                     // completion code has  been returned
                        error_handling(statusDataGlobal);
                        break;
                case VCI_WARNING:    // VCI_WARNING = A minor
                                     // error occured. The application may have to
                                     // perform error-handling processing depending on
                                     // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        }       // end switch

}       // end logout_function()
// End code example VCI_Logout
```

5.11 Terminating a VALUES Session

The VCI_Disconnect entry point is used by the application to disconnect from VALUES.

```
// code example VCI_Disconnect
#include “Values.h”
#include “ehs_konst.h”

// global definitions: the stack persistency is not enough for these variables as they must be used across multiple
// application functions
int     myConnectionID;     // myConnectionId identifies the session and
                            // is mandatory input for all future interface calls
void disconnect_function()
{
        // define variables in VCI-format
        ReqCntrlT           reqControl;
        DiscnctReqDataT     reqData;
        StatusDataT         statusDataGlobal;

        // fill the field ‘VCIver’ of reqControl
        strcpy(reqControl.VCIver, VCI_VERSION);     // set VALUES version for compatibility checking

        // at this point, the connection ID obtained as a returned parameter
        // from VCI_Connect is used to identify the session
        reqControl.connectionID = myConnectionID;

        // call VCI_Disconnect
        VCI_Disconnect      (&reqControl,
                        &reqData,
                        &statusDataGlobal );
```

```
        // Assess whether or not the call succeeded
        switch (statusDataGlobal.complSeverity)
        {
                case VCI_SUCCESS:    // VCI_SUCCESS = Successful completion
                        // add processing for successful login
                        break;
                case VCI_FATAL:      // VCI_FATAL = Fatal error has
                                     // occured. The application should perform a shutdown.
                        fatality_handling(statusDataGlobal);
                        break;
                case VCI_ERROR:      // VCI_ERROR = An error has
                                     // been detected. The application has to perform
                                     // error-handling processing depending on which
                                     // completion code has  been returned

                        error_handling(statusDataGlobal);
                        break;
                case VCI_WARNING:    // VCI_WARNING = A minor
                                     // error occured. The application may have to
                                     // perform error-handling processing depending on
                                     // which completion code has been returned.
                        warning_handling(statusDataGlobal);
                        break;
        }       // end switch

}       // end disconnect function
```

5.12 Setting up a Dispatch Loop using Galaxy

This section shows how to set up the application's Dispatch loop. The example is based upon Galaxy.

```
// code example Dispatch loop
void Connect_function(...)
{
        // this function calls VCI_Connect (see section 5.2), retrieves the VMQname,
        // and registers an event with callback for the Galaxy event loop which is then
        // the application's Dispatch loop

        // this function may be called by a Xetra® login window, passing the user ID and
        // password that a user has entered in this window

        // local variables
        int     myVMQ;

        // define and initiate variables in VCI-format (see section 5.2)

        // call VCI_Connect
        VCI_Connect (  &reqControl,
                       &reqData,
                       connectCallback,
                       &callbackContextData,
                       &respData,
                       &statusDataGlobal
                       );

        // Assess whether or not an error occured in the VCI_Connect call
        // error checking code may be entered here
        // for the example, the call is assumed successful

        // retrieve VMQName
        myVMQ = atoi(respData.VMQName);    // myVMQ now contains the file descriptor

        // register the event and a callback
        _ourVeventFD = teventFD::Register(myVMQ);
        _ourVeventFD->SetObserveReadProc(EDispatch);              // set callback for dispatch
        _ourVeventFD->SetObserveExceptProc(EException);   // set callback for exceptions
}       // end Connect_function
```

```
void EDispatch()
{
        // code to prepare for Dispatch call comes here



        // call VCI_Dispatch
        VCI_Dispatch   (&reqControl,
                        &statusDataGlobal );

        // error handling and cleanup code comes here
}       // end EDispatch
int main(int argc, char *argv[])
{
        // Code may be written here to initialise the application

        // Galaxy is initiated
        vstartup(argc, argv);

        // The Galaxy event loop is started
        vevent::Process();

        // More code may be entered here

        exit(EXIT_SUCCESS);
        return(EXIT_FAILURE);
}               // end main
```

5.13 Setting up a Dispatch Loop on the SUN Platform

This section shows how to retrieve the file descriptor of the VMQ and how to use it to set up the Dispatch loop. The example assumes a SUN platform.

```
// Note that this example does not compile: the example focuses on setting
// up the Dispatch loop, not on the VALUES calls. Programming examples of these
// VALUES entry points are in other sections of this chapter. Therefore,
// variable definition and initialisation, and definition of the callbacks
// have been omitted.

#include <poll.h>          // Library available on SUN platform.

#include "Values.h"
#include "ehs_konst.h"

int main ()
{
    // Local variables.
    struct pollfd pfd;     // This variable is used to poll the VMQ.
    int fd;                // fd is the file descriptor of the VMQ.

   // Define and initialise variables in VCI-format.

   // Call VCI_Connect.
    VCI_Connect( &reqControl,
            &reqData,
            connectCallback,
            &respData,
            &statusDataGlobal );
```

```
 // Assess whether or not the call succeeded.
switch (statusDataGlobal.complSeverity)
{
    case VCI_SUCCESS:       // VCI_SUCCESS = Successful completion.
        // Add processing for successful call.
        break;
    case VCI_FATAL:             // VCI_FATAL = Fatal error has occured.
                            // The application should perform a perform.
        fatality_handling(statusDataGlobal);
        break;
    case VCI_ERROR:                     // VCI_ERROR = An error has been detected.
                            // The application has to perform
                            // error-handling processing depending on which
                            // completion code has been returned.
        error_handling(statusDataGlobal);
        break;
    case VCI_WARNING:       // VCI_WARNING = A minor error occured. The
                            // The application may have to perform error-
                            // handling depending on which completion code
                            // has been returned.
        warning_handling(statusDataGlobal);
        break;
}   // end switch

// Get the file descriptor to the VMQ.
fd = atoi(respData.VMQname);

// Call VCI_Login.
VCI_Login( &reqControl,
           &reqData,
           loginCallback,
           &callbackContextData,
           &respData,
           &statusDataGlobal );
```

```
// Assess whether or not the call succeeded.
 switch (statusDataGlobal.complSeverity)
 {
    case VCI_SUCCESS:       // VCI_SUCCESS = Successful completion.
       // Add processing for successful call.
       break;
    case VCI_FATAL:         // VCI_FATAL = Fatal error has occured.
                            // The application should perform a perform.
       fatality_handling(statusDataGlobal);
       break;
    case VCI_ERROR:                 // VCI_ERROR = An error has been detected.
                            // The application has to perform
                            // error-handling processing depending on which
                            // completion code has been returned.
       error_handling(statusDataGlobal);
       break;
    case VCI_WARNING:       // VCI_WARNING = A minor error occured. The
                            // The application may have to perform error-
                            // handling depending on which completion code
                            // has been returned.
       warning_handling(statusDataGlobal);
       break;
 }  // end switch

// Call VCI_Submit.
 VCI_Submit( &reqControl,
       &reqData,
       responseCallback,
       &callbackContextData,
       &statusDataGlobal );
```

```
 // Assess whether or not the call succeeded.
switch (statusDataGlobal.complSeverity)
{
    case VCI_SUCCESS:       // VCI_SUCCESS = Successful completion.
        // Add processing for successful call.
        break;
    case VCI_FATAL:         // VCI_FATAL = Fatal error has occured.
                            // The application should perform a perform.
        fatality_handling(statusDataGlobal);
        break;
    case VCI_ERROR:                 // VCI_ERROR = An error has been detected.
                            // The application has to perform
                            // error-handling processing depending on which
                            // completion code has been returned.
        error_handling(statusDataGlobal);
        break;
    case VCI_WARNING:       // VCI_WARNING = A minor error occured. The
                            // The application may have to perform error-
                            // handling depending on which completion code
                            // has been returned.
        warning_handling(statusDataGlobal);
        break;
}   // end switch

 // Poll the VMQ.
// It is important to NOT read from the VMQ.
// The actual read is performed by VCI_Dispatch.

 pfd.fd = fd;
 pfd.events = 0;
 pfd.events = pfd.events | POLLIN;
 pfd.events = pfd.events | POLLRDNORM;
 pfd.events = pfd.events | POLLRDBAND;
 pfd.events = pfd.events | POLLPRI;
 pfd.events = pfd.events | POLLERR;

 do
 {
    if (poll(&pfd, 1, 0) < 0)
    {
        printf("\n Poll failed");
        break;
    }
 }
 while (pfd.revents == 0);
```

```
   // If a message is available, then call VCI_Dispatch.
    if ((pfd.revents & POLLERR) != 1)
    {
        // Call VCI_Dispatch.
        VCI_Dispatch(&reqCntrl, &statusDataGlobal);

        // Assess whether or not the call succeeded.
        switch (statusDataGlobal.complSeverity)
        {
        case VCI_SUCCESS:       // VCI_SUCCESS = Successful completion.
            // Add processing for successful call.
            break;
        case VCI_FATAL:             // VCI_FATAL = Fatal error has occured.
                                    // The application should perform a perform.
            fatality_handling(statusDataGlobal);
            break;
        case VCI_ERROR:                     // VCI_ERROR = An error has been detected.
                                    // The application has to perform
                                    // error-handling processing depending on which
                                    // completion code has been returned.
            error_handling(statusDataGlobal);
            break;
        case VCI_WARNING:       // VCI_WARNING = A minor error occured. The
                                    // The application may have to perform error-
                                    // handling depending on which completion code
                                    // has been returned.
            warning_handling(statusDataGlobal);
            break;
         }  // end switch
    } else
    {
        printf("\n Error on async. channel");
    }

    // The message has been successfully processed.
   // More code may be entered here.

    return(0);
}
```

6 Application Development

In this chapter an overview on the components of the development environment supporting the VALUES API call interface is given. Recommended compilers and linkers for each of the supported operating systems and language considerations are discussed.

6.1 Overview

The set of header files needed to develop an application using the Release 2 VALUES API is distributed by Deutsche Börse AG and is included with this document. These header files constitute a set to build an application on top of the VALUES API. However, as only the header files are distributed with this document, the application can be compiled, but not linked and is therefore not executable. The Release 2 Xetra® distribution disk distributed with a later document version will include the libraries needed to link and run the application.

Prerequisites for the VALUES API are described in chapter 7.

6.2 Development Environment

This section describes the usage of VALUES API libraries and include files to develop an end-user application on the supported platforms. Hints on compiling and linking with VALUES API components are given.

6.2.1 VALUES API Libraries

VALUES API libraries will be supplied with a later version of the Xetra® distribution disk. The disk will contain a library with the binary code of the VALUES API call interface. The library type (e.g., dynamic link library, sharable object, etc.) is dependent on the individual platform.

All libraries shipped with the Xetra® distribution disk must be linked with the application program. (See chapter 7 for listing of libraries.)

6.2.2 Include Files

For each of the supported platforms include files needed to use the VALUES API libraries are supplied with the Xetra® distribution disk. The required include files contain function prototypes, data structure definitions, and constant definitions. The include file syntax is ANSI C.

6.2.3 Makefile Templates

For each of the supported platforms a makefile template to support compilation and linking is provided with the Xetra® distribution disk.

6.2.4 Compilation

Detailed information on compiling with the VALUES API components is given in chapter 7. This information includes supported compilers, compiler settings, and platform specific details.

The following table gives the list of the recommended compiler for each supported platform:

Platform	Compiler	Compile options	
IBM AIX	IBM C Set++ for AIX (Version 3.1)	-i	include
		-c	1.compile 2.link
		-DAIX	AIX-specific
Sun Solaris	Sun C Compiler 3.0	-g	debugging information
		-w	no warnings
Microsoft Windows NT (Intel)	Visual C++ 4.X	none	

Table 6.1 : Recommended Compiler/Platform configurations

6.2.5 Linking

Linking considerations to integrate the VALUES API into the application is detailed here. The information includes linker settings and platform specific details. Be aware of that the application can be compiled, but can not be linked and executed with the distributed executables for VALUES API Release 1.

The following table gives the list of the recommended linker for each supported platform:

Platform	Linker	Link options
IBM AIX	IBM C Set++ for AIX (Version 3.1)	none
Sun Solaris	Sun C Linker 3.0	none
Microsoft Windows NT (Intel)	Visual C++ 4.X	none

Table 6.2 : Recommended Linker/Platform configurations

6.3 Language Considerations

The VALUES API call interface supports both C and C++ calling conventions. This means that applications developed in either C or C++ can use the VALUES API call interface. The VALUES API call interface itself uses ANSI C, the included files of the Xetra® distribution disk adapt to match this calling convention depending on the compiler used (i.e., native C compiler, native C++ compiler).

7 VALUES API Installation and Configuration (Development Environment)

In this chapter the VALUES API installation and configuration to develop an application using the VALUES API is discussed. The guide is split into three sections, one for each supported operating system.

7.1 Overview

VALUES consists of a set of components which need to be linked or accessed by the Front End application development. This set includes libraries, include-files and configuration files. The entire set of components (Xetra® distribution disk) will be delivered by Deutsche Börse.

Detailed installation steps are given in this chapter to support manual installation. The configuration of the three different versions was carried out in a way that yielded a similar installation process on the different platforms. Nevertheless, detailed information is given in each section to ensure successful implementation.

The VALUES API must be configured for each of the supported platforms. Detailed information on configuration options is given in sections 7.3.4, 7.4.4, and 7.5.4. Template configuration files are described in section 7.3.2.5 and delivered in electronic (ASCII) format with the VALUES API.

7.2 Supported Operating Systems

The VALUES API can be installed on the following operating systems:

Operating System	Version*	Comment
IBM AIX	4.1	
Sun Solaris	Generic 5.5.1	
Microsoft Windows NT (Intel)	4.0	Supported for Intel based hardware platforms only

* Versions may be subject to change

Table 7.3: Supported Operating Systems

7.3 IBM AIX

7.3.1 Installation Prerequisites

In order to properly install the VALUES API a number of prerequisites must be fulfilled.

7.3.1.1 Physical Storage and Memory Requirements

This section defines storage requirements including disk space needed to install the Xetra® distribution disk and memory requirements for the development hardware.

Disk Space (Mb)	50
Memory required (Mb)	64

7.3.1.2 Required Development Software

For C or C++ based application development a native C or C++ compiler is required to compile the include files with the interface declaration (function prototypes) and constant definitions. Also a native linker is required to link the VALUES libraries.

If you are not using the recommended compiler and linker (see chapter 6) your development tools must support:

- Calling externally defined C or C++ functions.
- Embedding C or C++ functions and exporting them so they can be called from VALUES (this is necessary to support application callbacks).

7.3.2 Installation Components

7.3.2.1 VALUES API Installation Components

The VALUES API installation components are grouped into the following categories:

- Libraries
- Include files
- Makefile templates
- Configuration template files

The following sub-sections give a list of all components per category.

7.3.2.2 Libraries

Names and location of the library are depending on the environment used (production or simulation). Libraries are also used in the Release 2 version of the VALUES API.

/usr/lpp/xetra/production/lib/libVALUES.a

/usr/lpp/xetra/simulation/lib/libVALUES.a

7.3.2.3 Include files

Names and location of the directory, depending on the environment (production or simulation), where include files are to be found:

/usr/lpp/xetra/production/values/include

/usr/lpp/xetra/simulation/values/include

The include files are:

Values.h	elbcode.h
app_rid.h	xrequest.h
ehs_konst.h	xbrdcast.h
requests.h	generalcommon.h
subject.h	privbcast.h
ehs_strukt.h	vld_val.h

7.3.2.4 Makefile templates

```
#for simulation
# VALUESBASE = /usr/lpp/xetra/simulation

#for production
VALUESBASE = /usr/lpp/xetra/production

SOURCE = $(VALUESBASE)/source
VALUES = $(VALUESBASE)/lib
VALUESBIN = $(VALUESBASE)/bin
DEBUG = -g
CFLAGS = -I$(VALUES) $(DEBUG)
CPPFLAGS = -I$(VALUES) $(DEBUG)
LDFLAGS = -bloadmap:lmap -L$(VALUES) -lValues -L$(VALUESBIN)

all: application_name

# rule example (see manual for proper use of make and rule syntax)
application_name: $(CC) -o /usr/lpp/xetra/simulation/bin/application_name $(LDFALGS)
        \
                -L /usr/lpp/xetra/simulation/source             -lapplication_lib\
                -L /usr/lpp/xetra/simulation/lib    -lvalues
```

7.3.2.5 Configuration template files

System Xetra® configuration file:

/usr/lpp/xetra/simulation/cfg/xetrasys.ini

/usr/lpp/xetra/production/cfg/xetrasys.ini

Member Xetra® configuration file:

/usr/lpp/xetra/simulation/cfg/xetrambr.ini

/usr/lpp/xetra/production/cfg/xetrambr.ini

Remote Workstation Xetra® configuration file:

/usr/lpp/xetra/simulation/cfg/xetrarws.ini

/usr/lpp/xetra/production/cfg/xetrarws.ini

7.3.3 Installation Steps

The installation steps to be followed when installing the Xetra® distribution disk are contained in the Installation Guide.

7.3.4 Configuration Options

For the configuration options, refer to the Front End Operations Guide.

7.4 Sun Solaris

7.4.1 Installation Prerequisites

In order to properly install the VALUES API a number of prerequisites must be fulfilled.

7.4.1.1 Physical Storage and Memory Requirements

This section defines storage requirements including disk space needed to install the Xetra® distribution disk and memory requirements for the development hardware.

Disk Space (Mb)	50
Memory required (Mb)	64

7.4.1.2 Required Development Software

For C or C++ based application development a native C or C++ compiler is required to compile the include files with the interface declaration (function prototypes) and constant definitions. Also a native linker is required to link the VALUES libraries.

If you are not using the recommended compiler and linker (see chapter 6) your development tools must support:

- Calling externally defined C or C++ functions.
- Embedding C or C++ functions and exporting them so they can be called from VALUES (this is necessary to support application callbacks).

7.4.2 Installation Components

7.4.2.1 VALUES API Installation Components

The VALUES API installation components are grouped into the following categories:

- Libraries
- Include files
- Makefile templates
- Configuration template files

The following sub-sections give a list of all components per category.

7.4.2.2 Libraries

Names and location of the library are depending on the environment used (production or simulation). Libraries are also used in the Release 2 version of the VALUES API.

/opt/xetra/production/values/lib/libvalues.a

/opt/xetra/simulation/values/lib/libvalues.a

7.4.2.3 Include files

Names and location of the directory, depending on the environment (production or simulation), where include files are to be found:

/opt/xetra/production/values/include

/opt/xetra/simulation/values/include

The include files are:

Values.h	elbcode.h
app_rid.h	xrequest.h
ehs_konst.h	xbrdcast.h
requests.h	generalcommon.h
subject.h	privbcast.h
ehs_strukt.h	vld_val.h

7.4.2.4 Makefile templates

```
#for simulation
# VALUESBASE = /opt/xetra/simulation/

#for production
VALUESBASE = /opt/xetra/production/
SOURCE = $(VALUESBASE)/source
VALUES = $(VALUESBASE)/lib
VALUESBIN = $(VALUESBASE)/bin
DEBUG = -g
CFLAGS = -I$(VALUES) $(DEBUG)
CPPFLAGS = -I$(VALUES) $(DEBUG)
LDFLAGS = -bloadmap:lmap -L$(VALUES) -lValues -L$(VALUESBIN)

all: application_name

# rule example (see manual for proper use of make and rule syntax)
application_name: $(CC) -o /opt/xetra/simulation/bin/application_name $(LDFALGS)    \
                  -L /opt/xetra/simulation/source          -lapplication_lib\
                  -L /opt/xetra/simulation/lib             -lvalues
```

7.4.2.5 Configuration template files

System Xetra® configuration file:

/opt/xetra/simulation/cfg/xetrasys.ini

/opt/xetra/production/cfg/xetrasys.ini

Member Xetra® configuration file:

/opt/xetra/simulation/cfg/xetrambr.ini

/opt/xetra/production/cfg/xetrambr.ini

Remote Workstation Xetra® configuration file:

/opt/xetra/simulation/cfg/xetrarws.ini

/opt/xetra/production/cfg/xetrarws.ini

7.4.3 Installation Steps

The installation steps to be followed when installing the Xetra® distribution disk are contained in the Installation Guide.

7.4.4 Configuration Options

For the configuration options, refer to the Front End Operations Guide.

7.5 Microsoft Windows NT (Intel)

7.5.1 Installation Prerequisites

In order to properly install the VALUES API a number of prerequisites must be fulfilled.

7.5.1.1 Physical Storage and Memory Requirements

This section defines storage requirements including disk space needed to install the Xetra® distribution disk and memory requirements for the development hardware.

Disk Space (Mb)	50
Memory required (Mb)	64

7.5.1.2 Required Development Software

For Windows NT the only supported Compiler is Visual C++ 4.X (see chapter 6).

7.5.2 Installation Components

7.5.2.1 VALUES API Installation Components

The VALUES API installation components are grouped into the following categories:

- Libraries
- Include files
- Configuration template files

The following sub-sections give a list of all components per category.

7.5.2.2 Libraries

Names and location of the library are depending on the environment used (production or simulation). Libraries are also used in the Release 2 version of the VALUES API.

[L:]\opt\xetra\production\lib\values.lib

[L:]\opt\xetra\simulation\lib\values.lib

7.5.2.3 Include files

Names and location of the directory, depending on the environment (production or simulation), where include files are to be found:

[L:]\opt\xetra\production\values\include

[L:]\opt\xetra\simulation\values\include

The include files are:

Values.h	elbcode.h
app_rid.h	xrequest.h
ehs_konst.h	xbrdcast.h
requests.h	generalcommon.h
subject.h	privbcast.h
ehs_strukt.h	vld_val.h

7.5.2.4 Makefile templates

The NT(Intel) Visual C++ development environment contains the so-called Microsoft Developer Studio, which is capable of automatically building the makefiles. Thus no makefile templates are needed.

7.5.2.5 Configuration template files

System Xetra® configuration file:

[L:]\opt\xetra\simulation\cfg\xetrasys.ini

[L:]\opt\xetra\production\cfg\xetrasys.ini

Member Xetra® configuration file:

[L:]\opt\xetra\simulation\cfg\xetrambr.ini

[L:]\opt\xetra\production\cfg\xetrambr.ini

Remote Workstation Xetra® configuration file:

[L:]\opt\xetra\simulation\cfg\xetrarws.ini

[L:]\opt\xetra\production\cfg\xetrarws.ini

7.5.3 Installation Steps

The installation steps to be followed when installing the Xetra® distribution disk are contained in the Installation Manual.

8 Attachment 1 : Functional Overview/Background

The following is a suggested list of sources of functional information concerning Xetra® Release 2 :

- IBIS-R LU 6.2-Interface Description Version 1.4 as of 24-APR-1997
- IBIS-R User Manual Bond Trading Manual Version 1.4 as of 24-APR-1997

9 Attachment 2: VALUES API Completion Codes

The VALUES API call interface communicates status information with a completion code data field. Each entry point generates a number of different completion codes. Some codes are shared among the entry points.

The table below shows from which Xetra® components (Xetra® processes) exceptions can be returned in each VALUES API Call Interface entry point. The completion codes are a superset to the exception codes listed in Chapter 3 VALUES API Call Interface Reference.

	Session Manager	Availability Manager	LAN Transport Managers	Security Manager	Back End Subsystem	Back End
VCI_Connect	X		X	X		
VCI_Disconnect	X					
VCI_Login	X		X		X	X
VCI_Logout	X		X		X	X
VCI_Subscribe	X					
VCI_Unsubscribe	X					
VCI_Submit	X		X		X	X[1]
VCI_Dispatch						
Application Callback	X		X		X	X

Table 9.1 Xetra® system components exceptions returned in VALUES API entry points

Following is a short description of each Xetra® system component.

Session Manager

The Session Manager provides a „fire wall“ between external application processes and the Xetra® Front End. Applications connect to the Xetra® Front End via the Session Manager which authenticates and manages the connection.

Availability Manager

The Availability Manager is responsible for startup and shutdown of Xetra® Front End processes.

[1] depending on the application request

Broadcast Receiver

The Broadcast Receiver is responsible for receiving broadcasts from the Republisher.

Republisher

The Republisher is responsible for forwarding broadcasts, arriving from the Deutsche Börse Applications, to the Broadcast Receivers.

WS LAN Transport Manager

The WS (workstation) LAN Transport Manager is responsible for the transport of request/response messages between the WS and the MISS.

MISS LAN Transport Manager

The MISS LAN Transport Manager is responsible for the transport of request/response messages between the MISS and the WS.

Front End Security Manager

The Front End Security Manager is responsible for authorization of connections to the Xetra® Front End.

Back End Specific Subsystem

The Back End Specific Subsystem for IBIS-R consists of a "BESS Manager" (including “Cache”), "SNA Session Handlers" and the "SNA Server". The BESS Manager handles the transformation of messages from the Xetra® message to the IBIS-R format and back. It passes converted messages to the appropriate SNA Session Handler.

The SNA Session Handler is responsible for allocating and deallocating SNA conversations with the Deutsche Börse Applications. To do so, it uses services of the SNA Server. The SNA Server establishes and monitors connections with the Deutsche Börse Applications, transfers requests and captures responses.

The Back End Specific Subsystem for Xetra® consists of a "BESS Xetra® Manager", "Broadcast Server", "Broadcast Retransmitter", and the "Xetra® WAN Transport Manager". The BESS Xetra® Manager is the entry point to the BESS Xetra® from a front end point of view, and performs all application related message processing within the BESS.

The Broadcast Server is responsible for subscribing to data streams on the WAN, capturing them via the BESS Xetra®'s WAN Transport Manager, and forwarding them to the Republisher.

The Broadcast Retransmitter is responsible for handling retransmission requests for private recoverable broadcasts passed on by the Applications using the VALUES API.

The Xetra® WAN Transport Manager is the entry point to the BESS Xetra® from a back end point of view, performs message conversion, and performs the communication with the Communication Server.

The following tables enlist all valid codes, their value and corresponding text. The tables are sorted by completion code within each Xetra® system component.

9.1 General Exception Codes

The exceptions listed here may be generated by any Xetra® Front End component.

Exception Code	Exception Message	Description
00000	SUCCESSFUL COMPLETION	N/N
00001	ASSERTION FAILED	A precondition of a function was not fulfilled.
00002	STARTUP SUCCESSFUL	Completion code after each successful process startup.
00003	SHUTDOWN IN PROGRESS	The Xetra® Front End was in the shut down state and no more messages were sent to other processes.
00004	INVALID ENTRY IN MSG.MASTER	Incomplete or damaged exception message file.
00010	AN OPERATING SYSTEM CALL FAILED	Internal program error. Report to help desk.
00012	INVALID REQUEST ID	Internal program error. Report to help desk.
00013	INVALID MESSAGE TYPE (SHOULD BE REQ/RESP, BROADCAST OR CONTROL)	Internal program error. Report to help desk.
00014	OUT OF MEMORY	Lack of memory resources. The process could not allocate the required memory.
00015	INVALID VERSION NUMBER	The Xetra® Front End version does not match the VALUES API version.
00016	FUNCTION OF NOT INITIALIZED MODULE CALLED	Internal program error. Report to help desk.
00017	MODULE CANNOT BE INITIALIZED	Internal program error. Report to help desk.
00018	INVALID OPERATION	Internal program error. Report to help desk.

Exception Code	Exception Message	Description
00019	CHANNEL IS CLOSED	A inter-process communication channel was closed unexpectedly.
00020	LISTENING CHANNEL IS CLOSED	N/N
00021	OUTPUT QUEUE LIMIT IS REACHED	The channel queue size set with the QueueSize parameter within the InterProcessCommunication section in the Xetra® Configuration File has been exceeded.
00022	INVALID FUNCTION CALL	Internal program error. Report to help desk.
00023	PARAMETER CHECK FAILED	Internal program error. Report to help desk.
00024	NO PARAMETERS NEEDED IN STARTING THIS PROCESS	N/N
00031	CHILD PROCESS COULD NOT BE STARTED	There was not enough memory to start up a further child process or the process number limit was reached.
00035	COULD NOT GET SYSTEM TIME	Internal program error. Report to help desk.
00037	THE CONNECT MESSAGE BODY HAS THE WRONG LENGTH	Internal program error. Report to help desk.
00042	WRITE WILL BE COMPLETED ASYNCHRONOUSLY	N/N
00043	COULD NOT OPEN CHANNEL - PEER PROCESS MAY BE DOWN	N/N
00044	READ ERROR OSAIPC	A read exception on a communication channel occurred.
00045	WRITE ERROR OSAIPC	A write exception on a communication channel occurred.
00046	COULD NOT CREATE LISTEN - PORT IN USE	Indicates that a process was trying to use a listener port already in use. Either the same process is already running or another process listens on the port.

Exception Code	Exception Message	Description
00047	MESSAGE TO BE READ FROM CHANNEL IS TOO LONG	N/N
00048	ERROR SCANNING HASH TABLE	Internal program error. Report to help desk.
00049	ERROR CREATE LISTEN - EMPTY PORT	N/N
00050	VALUE TO BE SET IN A MSG FIELD IS TOO BIG TO FIT IN THE FIELD	Internal program error. Report to help desk.
00069	INVALID USER OR PASSWORD	N/N
00071	INVALID CONFIGURATION FILE ENTRY	An entry in the Xetra® Configuration File does not exist or is out of range.
00072	END OF FILE	Internal program error. Report to help desk.
00073	FILE TRUNCATED	N/N
00074	INVALID FILE TYPE	N/N
00075	CANNOT OPEN FILE	The File could not be opened because it does not exist or has no read permission.
00076	A CONFIGURATION FILE ENTRY HAS A WRONG LENGTH	N/N
00077	A STANDARD FUNCTION FAILED	The read() function failed.
00078	A STANDARD FUNCTION FAILED	The write() function failed.
00079	A STANDARD FUNCTION FAILED	The fseek() function failed.
00080	A STANDARD FUNCTION FAILED	The ftell() function failed.
00081	A STANDARD FUNCTION FAILED	The flush() function failed.
00082	INVALID RECORD SIZE	N/N

Exception Code	Exception Message	Description
00083	RECORD NOT IN USE	N/N
00302	TOO MANY ARGUMENTS	N/N
00303	NOT ENOUGH ARGUMENTS	N/N
00304	WRONG ARGUMENT (C OR V)!	N/N
00305	CHECKSUM IS INCORRECT	The message may have been sent or changed without using the standard Xetra® Front End security mechanism.
00306	CHECKSUM IS CORRECT	N/N

9.2 Session Manager

Exception Code	Exception Message	Description
00025	INVALID CONNECTION ID	Invalid connection ID received by the Session Manager. Possibly an unauthorized attempt to connect to the Xetra® Front End (i.e., bypassing VALUES API
00026	INVALID SEQUENCE NUMBER	N/N
00027	CHANNEL NOT ESTABLISHED	N/N
00028	UNKNOWN SUBSCRIPTION ID	N/N
00029	CANNOT DELIVER BROADCAST MESSAGE (EMPTY CLOSURE)	No registered recipient of the message could be found.
00030	RECEIVED REPEATED CONNECT ON ONE CHANNEL	N/N
00038	MAXIMUM NUMBER OF CONNECTIONS REACHED	N/N

9.3 Security Manager

Exception Code	Exception Message	Description
00060	USER IS NOT ALLOWED TO USE XETRA OR USER IS NOT REGISTERED	N/N
00061	PASSWORD IS NOT OK FOR XETRA USER	N/N
00062	NT: DOMAIN PASSED DOES NOT EXIST	Windows NT (Intel) only
00063	NT: RETURNED A UNKNOWN OPERATION SUBSYSTEM ERROR	Windows NT (Intel) only
00064	NT: A PARAMETER WAS EITHER INVALID OR HAD INVALID CHARACTERS	Windows NT (Intel) only
00065	PROVIDED HOST NAME UNKNOWN	N/N

9.4 Availability Manager

Exception Code	Exception Message	Description
00090	ARCHITECTURE STARTUP COMPLETED	Startup completed of 1 architecture process or all architecture processes.
00091	PROCESS STARTUP TIMEOUT	Unable to start process within specified timeout.
00092	ARCHITECTURE SHUTDOWN	Availability Manager decides to shut down architecture.
00093	ARCHITECTURE SHUTDOWN REQUEST RECEIVED	Availability Manager received request to shut down architecture.
00094	ARCHITECTURE PROCESS HAS TERMINATED	N/N
00095	TRYING TO START PROCESS	An architecture process is being started.
00097	ENFORCE TERMINATION	N/N

Exception Code	Exception Message	Description
00444	SERVICE AVAILABLE	N/N

9.5 Back End Subsystem

9.5.1 IBIS-R specific

The following table lists exceptions generated by the architecture level of the IBIS-R BESS.

Exception Code	Exception Message	Description
00101	CONVERSATION FAILED	SNA open or allocate function failed.
00102	INVALID UID OR LOGGED IN VIA LU2	The SNA receive and wait function failed or an invalid user ID and/or password was given.
00103	CACHE CONVERSION FAILED	A wrong wkn was passed.
00104	TRADER NOT LOGGED IN	Submit to IBIS-R carried out without logging in before.
00105	TRADER ALREADY LOGGED IN	Already logged on to IBIS-R.
00106	INTERNAL ERROR: SESSION HANDLER	Internal program error. Report to help desk.
00107	INTERNAL ERROR: STORED MESSAGE NOT FOUND	Internal program error. Report to help desk.
00108	CONVERSATION TIMEOUT: MESSAGE DISCARDED	N/N
00109	CONVERSATION TIMEOUT: SESSION HANDLER DOWN	N/N
00110	ARCHITECTURE SHUTDOWN WITH ERROR	The SNA close function failed.
00111	NOT AVAILABLE UNTIL MARKET DATA LOAD COMPLETED	The user tried to logon to IBIS-R, but the market load phase was not finished.
00113	SUCCESSFUL IBIS-R STARTUP	N/N
00115	IBIS-R SYSTEM NOT AVAILABLE ANY MORE	N/N
00116	BACK END MANAGER STARTED SUCCESSFULLY	N/N
00117	BACK END MANAGER	N/N

Exception Code	Exception Message	Description
	STOPPED SUCCESSFULLY	
00118	SEND QUEUE FULL	N/N

The following table lists messages generated by the Cache component of the IBIS-R BESS architecture level.

Exception Code	Exception Message	Description
04121	INVALID COUNTER	The value of the ctr parameter was invalid. Counter identifies the news (information from the Deutsche Börse AG) for which the text is to be returned.
04122	INVALID DEPTH	The value of the dep parameter was invalid. Depth, indicates how many data records should be returned at most. If fewer records exist, only those available are returned.
04123	INVALID DIALOG TYPE	The value of the dlgTyp parameter was invalid. Dialog type of corresponding IBIS message.
04124	INVALID XETRA MESSAGE BODY	The value of the message body of the Xetra® request message was invalid.
04125	INVALID INQUIRY TYPE	The value of the inqTyp parameter was invalid.
04126	INVALID INSTRUMENT SUBTYPE	The value of the instSubTyp parameter was invalid.
04127	INVALID MAXIMUM INTEREST RATE	The value of the intRatMax parameter was invalid. Maximum interest rate identifies the maximum interest rate for instruments that should be returned with an instrument inquiry.
04128	INVALID MINIMUM INTEREST RATE	The value of the intRatMin parameter was invalid. Minimum interest rate identifies the minimum interest rate for instruments that should be returned with an instrument inquiry.
04129	INVALID ISSUER MNEMONIC	The value of the issrMnem parameter was invalid. Mnemonic name of issuer identifies the issuer of instruments that should be returned with an instrument inquiry.
04130	INVALID MAXIMUM MATURITY DATE	The value of the mrtyDatMax parameter was invalid. Maximum maturity date identifies the earliest maturity date for instruments that should be returned with an instrument inquiry.
04131	INVALID MINIMUM MATURITY DATE	The value of the mrtyDatMin parameter was invalid. Minimum maturity date identifies the earliest maturity date for instruments that should

Exception Code	Exception Message	Description
		be returned with an instrument inquiry.
04132	INVALID REQUEST ID	The value of the reqId parameter was invalid. Xetra® request ID required by the cache to map requests and responses. The reqId is binary and fixed 12 bytes long..
04133	INVALID SCOPE	The value of the scp parameter was invalid. Scope indicates whether all offers, own member offers or own offers are inquired.
04134	INVALID TRADER ID	The value of the trdId parameter was invalid.
04135	INVALID WKN	The value of the wkn parameter was invalid.
04136	INVALID DISPOSITION INDICATOR	The value of the dspoInd parameter was invalid. Disposition indicator identifies the disposition for instruments that should be returned with an instrument inquiry.
04137	INVALID MINIMUM TIME	The value of the timeMin parameter was invalid. Minimum time specifies the minimum in the selection parameter time spread.
04138	INVALID MAXIMUM TIME	The value of the timeMax parameter was invalid. Maximum time specifies the maximum in the selection parameter time spread.
04140	CONVERSION ERROR	The conversion from the back end application to the appropriate dialog was unsuccessful.
04150	SYSTEM ERROR	A system call, e.g. malloc, failed.
04160	FATAL ERROR	The process was terminated.
04219	DATA NOT FOUND	An inquiry returned no data.

9.5.2 XETRA® specific

9.5.2.1 XETRA® BESS Architecture

Exception Code	Exception Message	Description
00216	REFERENCE FILE CHECK FAILED	Version of reference file is wrong.
00450	THE RECIPIENT PROCESS NAME IS NOT KNOWN	N/N
00451	UNKNOWN MESSAGE TYPE	N/N
00460	NO BUSINESS DATE IN REFERENCE DATA FILE	N/N
00461	NO MISS IN REFERENCE DATA FILE	The MISS can't find its own node number in the reference data file.
00462	INVALID RECORD IN CHECK REFERENCE DATA FILE	Invalid contents of reference data file.
00470	DUPLICATE TRANSACTION CODE	Internal architecture error in compression/encryption transaction code table.
00471	NO TRANSACTION CODE TABLE CREATED	Internal architecture error in compression/encryption transaction code table.
00472	NO TRANSACTION CODE WRITTEN TO TABLE	Internal architecture error in compression/ encryption transaction code table.
00473	COMPRESSION FAILED	N/N
00474	DECOMPRESSION FAILED	N/N
00475	ENCRYPTION FAILED	Reserved for Release 3.
00476	DECRYPTION FAILED	Reserved for Release 3.
00477	COMPRESSION OR ENCRYPTION FAILED	N/N
00478	DECOMPRESSION OR DECRYPTION FAILED	N/N
00479	MESSAGE TOO LONG FOR COMPRESSION/ENCRYPTION	N/N
00480	MESSAGE TOO LONG FOR	N/N

Exception Code	Exception Message	Description
	DECOMPRESSION/DECRYPTION	
00500	NO CONNECTION TO CS	N/N
00501	REQUEST CACHE FULL	N/N
00502	MESSAGE COMMAND UNKNOWN	N/N
00503	INVALID XERVICE ID	N/N
00505	CONVERSION BETWEEN FE AND BE MESSAGE FAILED	N/N
00506	INVALID SERVICE NOTIFICATION	N/N
00517	NO APPLICATION PROCESS FOUND FOR PROVIDED REQUEST ID	N/N
00518	NO LOGIN WHILE BESS UNAVAILABLE	N/N
00520	MISS ALREADY CONNECTED	N/N
00521	NO CS FOUND IN REFERENCE FILE	N/N
00522	CONNECTION TO CS LOST	N/N
00598	BACK END APPLICATION ERROR	N/N
00599	BACK END ARCHITECTURE ERROR	N/N
00650	SEQUENCE NUMBER FOR RETRANSMISSION REQUEST HAS NOT BEEN SENT YET	Application could not have received a ROB with the requested or a higher sequence number.
00651	SEQUENCE NUMBER FOR RETRANSMISSION REQUEST NOT AVAILABLE YET	MISS is recovering the requested broadcast messages.
00652	NO RESPONSE ON RETRANSMISSION REQUEST DUE TO INTERNAL ARCHITECTURE ERROR:	Broadcast Retransmitter.
00653	REQUESTED STREAM	Broadcast Retransmitter.

Exception Code	Exception Message	Description
	UNKNOWN TO RETRANSMITTER	
00654	INVALID SEQUENCE NUMBERS OR KEYDATCTRLBLC IN REQUEST	Broadcast Retransmitter.
00655	ERROR ON OPENING STATUS OR LOG FILES	Broadcast Retransmitter.
00656	CHANNEL LOST TO BESS XETRA MANAGER	Broadcast Retransmitter.

9.5.2.2 XETRA® BESS Application

Exception Code	Exception Message	Description
04500	INSTRUMENT DOES NOT EXIST	The request can not be performed, because the instrument does not exist.
04501	INSTRUMENT IS NOT ACTIVE	The request can not be performed, because the instrument is not valid.
04502	MEMBER ID IS NOT EXCHANGE	The requesting member is not allowed to perform this request.
04503	FIELD CONTAINS NON PRINTABLE CHARACTERS	N/N
04504	INVALID BASIC ORDER TYPE	Entered Order Type is not valid. Valid entries are M, L depending on the limit entered. If the limit is zero it must be M.
04505	INVALID BUY/SELL INDICATOR	Buy/Sell indicator is not valid. Valid entries are either B, S or space
04506	INVALID QUOTE QUANTITY	Entered quote quantity is not a valid number or is not equal to round lot quantity.
04507	INVALID LIMIT	Entered limit price is not a valid number or entered limit price failed tick-size validation.
04508	INVALID MEMBER INTERNAL ORDER NUMBER	An entered member-internal order number failed syntax validation or an entered order number is not numeric.

Exception Code	Exception Message	Description
04509	INVALID EXPIRY DATE	Entered expiry date is not a valid date or is not greater or equal to today's date or is not less or equal to today + one year or is not equal to today's date while execution restriction code is F or I.
04510	INVALID ORDER QUANTITY	Entered order quantity is not a valid number or is not equal to round lot quantity.
04511	INVALID USER ID	User ID failed syntax check.
04512	INVALID ACCOUNT TYPE	Account type is not equal to A, P or B.
04513	INVALID EXECUTION RESTRICTION TYPE	Execution restriction type is not equal to FOC, IOC or space.
04515	INVALID LIMIT PRICE BUY	Entered limit price is not a valid number or entered limit price failed tick-size validation.
04516	INVALID LIMIT PRICE SELL	Entered limit price is not a valid number or entered limit price failed tick-size validation.
04517	INVALID TRADING RESTRICTION TYPE	Entered trading restriction is not valid for trading model type. If the trading model type for an instrument is auction only or multiple auction then the trading restriction must be set to none. If the trading model is continuous trading with auctions the
04518	INVALID DATE	Entered date is not a valid date or entered min date is greater than entered max date.
04519	INVALID TIME	Entered time is not a valid time or entered min time is greater than entered max time.
04520	INVALID BUY/SELL LIMIT	Check if limit price buy is less than limit price sell failed.
04521	INVALID FILTER LIMIT PRICE	Entered price failed syntax validation or entered price failed tick-size

Exception Code	Exception Message	Description
		validation or entered min price is greater than entered max price.
04522	INVALID ORDER NUMBER	Order number is not numeric.
04523	INVALID TRADER ID	Member branch ID failed syntax validation or member institution ID failed syntax validation or part.-subgroup failed syntax validation or part.-number failed syntax validation or entered part.-subgroup is not equal to subgroup of current user.
04524	INVALID ACCOUNT NUMBER	Account number is not equal to 1.
04525	SELL/BUY QUANTITY NOT EQUAL	Quote quantity sell is not equal to quote quantity buy.
04526	ERROR IN LOADING INSTRUMENT REFERENCE DATA	Instrument reference data could not be loaded.
04527	INVALID KEY DATA CONTROL BLOCK	Key for next inquiry is corrupt.
04528	INVALID INSTRUMENT LIST REQUEST	N/N
04529	INVALID TRANSACTION ID	N/N
04531	INVALID COUNTERPARTY	The entered Counterparty is not a valid User ID.
04532	INVALID MEMBER ID	The entered Member ID is not a valid Member ID of the Xetra® system.
04533	INVALID CLEARING MEMBER ID	The entered Clearing Member ID is either not a valid Member ID of the system or the Member is no Clearing Member.
04534	INVALID PRICE CURRENCY CODE	The entered Price Currency Code is not valid.
04535	INVALID EXCHANGE MEMBER	N/N
04536	INVALID QUOTE SPREAD	The entered Quote Spread is greater than the maximum spread allowed for that instrument.
04537	INVALID REQUEST DATA	General error code (request message

Exception Code	Exception Message	Description
		body corrupted etc.).
04538	INVALID MODIFICATION OF ACCOUNT TYPE	The entered Account Type is either no valid Account Type or the modification itself is not allowed. Valid Account Types are: „A“, „B“ or „P“. Only modification from „A“ to „P“ and vice versa are allowed.
04539	DATE REQUIRED	N/N
04540	BEGIN TIME GREATER END TIME	N/N
04541	INVALID RESOURCE ACCESS LEVEL	The entered Resource Access Level are either not known by the system or not valid for the specific User ID. The Resource Access Level of a User can not exceed the Resource Access Level of its member.
04542	INVALID AGENT INDICATOR	N/N
04543	INVALID SENIOR INDICATOR	N/N
04544	INVALID PROPRIETARY INDICATOR	N/N
04545	INVALID BETREUER INDICATOR	N/N
04546	ERROR READING TABLE01 - NO BUSINESS DATE FOUND	Business date not found.
04547	INVALID PRICE REASONABILITY FLAG	Entered flag is not equal to Y or N.
04549	INVALID RESPONSE DATA	General error code (response message body corrupted etc.).
04550	INVALID NEWS SUBJECT	The entered News Subject is not known by the system.
04551	INVALID NEWS MESSAGE	The entered News Message is not valid.
04552	INVALID MAXIMUM ORDER QUANTITY	The entered Maximum Order Quantity is not valid.
04553	INVALID TRADER NAME	The entered Trader ID is not valid.

Exception Code	Exception Message	Description
04560	INVALID REPORT SELECTION INDICATOR	This error code is returned when a member tries to select/de-select a report and the selection indicator is invalid.
04561	INVALID TEXT	The text field contains invalid characters.
04562	INVALID PASSWORD	The new password is syntactical invalid.
04571	INVALID COMBINATION OF ORDER RESTRICTIONS	The combination of order restrictions is invalid.
04572	EXECUTION RESTRICTIONS ARE ONLY VALID WITH ORDERS GFD	Execution restrictions can only be specified for orders valid the same business day.

9.6 Back End Application

9.6.1 IBIS-R

The table below contains two distinct exception codes. The IBIS-R Exception Code is generated by IBIS-R and attached to the exception message before it is logged and displayed by the end-user application. Use the IBIS-R exception code to find more information about the exception in your IBIS-R manuals.

The Xetra® Front End exception code is logged only and not displayed by the end-user application.

Xetra® Front End Exception Code	IBIS-R Exception Code	Exception Message
03999	3999	UNDEFINED IBIS-R ERROR
09796	2294	IBIS-R INTERNAL: IMS ERROR
09798	2296	IBIS-R INTERNAL: RESOURCE PROBLEMS
09799	2297	IBIS-R IN SHUT DOWN PROCESS
097CD	2397	INSTRUMENT CANNOT BE LOADED
09801	2601	ENTRY ERROR: INVALID USER-ID
09802	2602	ENTRY ERROR: INVALID PASSWORD
09803	2603	USER-ID NOT FOUND IN ACCESS CONTROL DATABASE
09804	2604	PASSWORD INVALID
09805	2605	USER-ID VALIDITY PERIOD EXPIRED; INFORM SECURITY OFFICER
09806	2606	PASSWORD OF FIRST LOGON; ENTER NEW PASSWORD
09807	2607	PASSWORD VALIDITY PERIOD EXPIRED
09808	2608	ACCESS-AUTHORIZATION SEGMENT MISSING; INFORM SECURITY OFFICER
09809	2609	ACCESS DENIED; AUTHORIZATION IS MISSING; INFORM SECURITY OFFICER
09810	2632	NEW PASSWORD INVALID
09811	2633	NEW PASSWORD MUST DIFFER FROM OLD PASSWORD
09901	3201	ERROR FIELD: INSTRUMENT ID INVALID OR NOT ENTERED

Xetra® Front End Exception Code	IBIS-R Exception Code	Exception Message
09902	3202	ERROR FIELD: INSTRUMENT NAME INVALID
09906	3206	ERROR FIELD: MEMBER ID INVALID OR NOT ENTERED
09907	3207	ERROR FIELD: MEMBER ID OF COUNTERPARTY INVALID OR NOT ENTERED
09908	3208	ERROR FIELD: PRICE INVALID OR NOT ENTERED
09909	3209	ERROR FIELD: QUANTITY INVALID OR NOT ENTERED
09911	3233	ERROR FIELD: DIFFERING TRADE DATE CANNOT BE IN THE FUTURE
09912	3234	MAXIMUM NUMBER OF INSTRUMENTS IN WHICH ONE MEMBER CAN ENTER OFFERS IS REACHED
09913	3235	ERROR FIELD: SETTLEMENT DATE IS INVALID
09915	3237	ERROR FIELD: DIFFERING TRADE DATE WAS NO WORKING DAY FOR THE BUYER
09916	3238	ERROR FIELD: DIFFERING TRADE DATE WAS NO WORKING DAY FOR THE SELLER
09918	3240	FUNCTION WITHIN TEST ENVIRONMENT IS NOT AVAILABLE WITHOUT IMS
09919	3241	ERROR FIELD: SETTLEMENT DATE IS NO WORKING DAY FOR THE PURCHASER
09920	3264	ERROR FIELD: SETTLEMENT DATE IS NO WORKING DAY FOR THE SELLER
09923	3267	ERROR FIELD: USER REFERENCE IS INVALID OR NOT ENTERED
09925	3269	ERROR FIELD: TRADE TIME IS INVALID OR NOT ENTERED
09930	3296	ERROR FIELD: TRADE NUMBER IS INVALID OR NOT ENTERED
09931	3297	ERROR FIELD: TIME SELF IS INVALID OR NOT ENTERED
09932	3298	ERROR FIELD: TIME COUNTERPARTY IS INVALID OR NOT ENTERED
09933	3299	ERROR FIELD: TRADE TYPE IS INVALID OR NOT ENTERED
09935	3301	ERROR FIELD: AUTHORISATION IS INVALID OR NOT ENTERED

Xetra® Front End Exception Code	IBIS-R Exception Code	Exception Message
09937	3303	ERROR FIELD: CREATION TIME IS INVALID OR NOT ENTERED
09939	3305	PRESENT GROUP IS NOT NUMERIC
09940	3328	ERROR DURING CANCELLATION: NO TRADE FOUND TO BE DELETED
09941	3329	ERROR DURING CANCELLATION: NO BID/ASK FOUND TO BE DELETED OR CHANGED
09942	3330	ERROR DURING CANCELLATION: PURCHASE/SALE MUST BE DELETED BY THE COUNTERPARTY
09943	3331	ERROR DURING CANCELLATION: PURCHASE / SALE CAN BE DELETED ONLY
09944	3332	ERROR DURING CANCELLATION: IT IS NOT ALLOWED TO DELETE COUNTERPARTY ENTRY
09947	3335	IBIS-R INTERNAL: IMS IS NOT ACTIVE
09948	3336	PRESENT COUNTER IS NOT NUMERIC
09950	3360	ERROR FIELD: USER ID IS NOT VALID OR NOT ENTERED
09951	3361	ERROR FIELD: LL IS INVALID
09952	3362	ERROR FIELD: USER MODE IS INVALID
09953	3363	WRONG BUFFER: USED BUFFER GREATER THAN DEFINED SIZE
09954	3364	SAME INSTRUMENT CODE RECORDED TWICE
09955	3365	USER ID ALREADY IN USE
09956	3366	MEMBER ID ALREADY IN USE
09957	3367	IBIS-R INTERNAL: SAME TRADE RECORDED TWICE
09958	3368	NEW TRADE
09959	3369	TRADE
09960	3392	IBIS-R INTERNAL: BID/ASK PRESENTLY IN PROCESS; LALL/LUSR DENIED (REPETITION)
09961	3393	IBIS-R INTERNAL: LALL PRESENTLY IN PROCESS; REPEAT ENTRY

Xetra® Front End Exception Code	IBIS-R Exception Code	Exception Message
09962	3394	INPUT ERROR: DATE SYNTAX IS WRONG (DY25)
09963	3395	INPUT ERROR: DIFFERING TRADE DATE IS LATER THAN 90 DAYS
09964	3396	INPUT ERROR: DATE ENTERED IS NO WORKING DAY
09965	3397	DATE IS NOT NUMERIC
09967	3399	INPUT ERROR: POSTING CUT (SETTLEMENT DATE > NEXT WORKING DAY)
09968	3400	INPUT ERROR: SETTLEMENT CODE OR SETTLEMENT DATE MUST BE ENTERED
09970	3424	RESOURCE SHORTAGE: NO ENOUGH MEMORY
09971	3425	RESOURCE SHORTAGE: USER LIMIT IS REACHED
09974	3428	RESOURCE SHORTAGE: SCREEN SECURITY LIMIT IS REACHED
09975	3429	RESOURCE SHORTAGE: INSTRUMENT LIMIT IS REACHED
09976	3430	RESOURCE SHORTAGE: BID/ASK LIMIT IS REACHED
09977	3431	RESOURCE SHORTAGE: TRADE LIMIT IS REACHED
09978	3432	RESOURCE SHORTAGE: LU2 LIMIT REACHED
09979	3433	RESOURCE SHORTAGE: LU6.2 LIMIT REACHED
09980	3456	ERROR MESSAGE HEADER: FUNK-TYPE IS INVALID OR NOT ENTERED
09981	3457	ERROR MESSAGE HEADER: INFO-TYPE IS INVALID OR NOT ENTERED
09982	3458	IBIS-R INTERNAL: LOAD ERROR
09983	3459	IBIS-R INTERNAL: INDEX ERROR
09984	3460	IBIS-R INTERNAL: AT PRESENT NO TRADING POSSIBLE IN THIS INSTRUMENT
09985	3461	INPUT ERROR: NO INPUT OF SETTLEMENT CODE AND SETTLEMENT DATE
09986	3462	STATUS INVALID OR NOT ENTERED

Xetra® Front End Exception Code	IBIS-R Exception Code	Exception Message
09987	3463	INPUT ERROR: FOR STL. CODE 'I' THE DIFFERENT TRADE DATE COULD ONLY BE 3 DAYS PRIOR AT MAX.
09989	3465	INPUT ERROR: SETTLEMENT DATE MAY BE NO MORE THAN 90 DAYS IN THE FUTURE
09990	3488	INPUT ERROR: DATE IS NO WORKING DAY AT THE CLEARING BANK
09991	3489	IBIS-R INTERNAL: SETTLEMENT DATE IS INVALID OR NOT ENTERED
09994	3492	LOADING FINISHED
09995	3493	IBIS-R INTERNAL: INSTRUMENT IS NOT AVAILABLE ON DATABASE
099A1	3521	IBIS-R INTERNAL: INSTRUMENT IS NO FIXED INCOME
099A8	3528	YIELD IS NOT NUMERIC
099A9	3529	YIELD ENTRY NOT VALID
099AB	3531	LU2 INTERNAL:
099AD	3533	DELETION WAS UNSUCCESSFUL
099AE	3534	IBIS-R INTERNAL: LALL NOT POSSIBLE - PLS. DELETE ONE BY ONE
099AF	3535	IBIS-R INTERNAL: DURING THE PRE-TRADING PERIOD NO TRADING POSSIBLE
099AG	3536	IBIS-R INTERNAL: THIS INSTRUMENT IS CURRENTLY BLOCKED FOR TRADING
099AH	3537	INPUT ERROR: MINIMUM EXECUTION QUANTITY INVALID
099AI	3538	INPUT ERROR: PRICE CALCULATION FAILED
099AN	3543	IBIS-R INTERNAL: SELECTED BID / ASK IS NOT AVAILABLE
099AO	3544	IBIS-R INTERNAL: SUGGEST ALTERNATIVE BID / ASK
099AR	3547	SELECTED OFFER IS NOT AVAILABLE ANYMORE
099B4	3556	INPUT ERROR: NO TRADE WITH OWN MEMBER POSSIBLE
099B6	3558	OFFER IS NOT AVAILABLE ANYMORE, ALTERNATE OFFER IS SUGGESTED

Xetra® Front End Exception Code	IBIS-R Exception Code	Exception Message
099B7	3559	INPUT ERROR: BID HIGHER THAN BEST ASK
099B8	3560	INPUT ERROR: ASK LOWER THAN BEST BID
099BD	3565	IBIS-R INTERNAL: RESERVE RETURN CODE
099BE	3566	IBIS-R INTERNAL: RESERVE RETURN CODE
099BF	3567	IBIS-R INTERNAL: RESERVE RETURN CODE
099BG	3568	IBIS-R INTERNAL: INDEX ERROR
099BH	3569	TRADE TIME HAS TO BE ENTERED
099BI	3570	LU2 INTERNAL: NO OFFER WITHIN SELECTION
099BJ	3571	NATIONAL OR INTERNATIONAL SETTLEMENT DATE IS NOT ALLOWED FOR NEW ISSUES
099BM	3574	IBIS-R INTERNAL: STATUS OF INSTRUMENT IS INVALID
099BR	3579	INPUT ERROR: TEXT ENTRY CONTAINS WRONG CHARACTERS
099BS	3580	INPUT ERROR: LIMITS BIDS/ASKS PER MEMBER IN INSTRUMENT REACHED
099BT	3581	IBIS-R INTERNAL: UNEXPECTED COMBINATION OF OPTIONS
099BU	3582	ERROR FIELD: OFFER NUMBER IS WRONG
099BV	3583	IBIS-R INTERNAL: LIMIT BIDS/ASKS PER INSTRUMENT IS REACHED
099BW	3552	INPUT ERROR: QUANTITY TOO LARGE, CURRENT TRADING NUMBER EXCEEDS LIMIT
099BX	3553	IBIS-R INTERNAL: OFFER NUMBER DOUBLED
099BY	3554	IBIS-R INTERNAL: TEXT OF OFFER IS CHANGED
099BZ	3555	MINIMUM EXECUTION QUANTITY IS INVALID
099C0	3584	SETTLEMENT CODE IS INVALID
099C1	3585	IBIS-R INTERNAL: TRADE TYPE INVALID
099C2	3586	INSTRUMENT CANNOT BE LOADED - SECONDARY TRADING

Xetra® Front End Exception Code	IBIS-R Exception Code	Exception Message
099C3	3587	INSTRUMENT CANNOT BE LOADED - PRIMARY TRADING
099C4	3588	ISSUE IS NOT FROM AUTHORIZED USER
099C5	3589	FIRST ENTRY OF ISSUER CAN ONLY BE AN ASK OR A TELEPHONE SALE
099C6	3590	OFFER IS HELD BY ANOTHER MEMBER'S TRADER
099C7	3591	USER IS NO ISSUER
099C8	3592	IBIS-R INTERNAL:
099C9	3593	NO OFFER ENTRY POSSIBLE WHILE INSTRUMENT IS HELD
099CA	3594	RESOURCE SHORTAGE: USER LIMIT IS REACHED
099CB	3595	LU2 INTERNAL: NO OFFER ENTRY DURING MASS HOLD
099CC	3596	LU2 INTERNAL:
099CD	3597	INSTRUMENT CANNOT BE LOADED
099CE	3598	INTERNAL OPERATOR MESSAGE
099CF	3599	LU2 INTERNAL: NOT HELD OFFERS CANNOT BE RELEASED
099CG	3600	SETTLEMENT DATE HAS TO BE GREATER FOR NEW ISSUES
099CH	3601	LU2 INTERNAL: SELECTION ALREADY EXISTS
099CI	3602	LU2 INTERNAL: NO INSTRUMENT IN SELECTION
099CJ	3603	LU2 INTERNAL: SELECTION CRITERIA INVALID

9.6.2 XETRA®

Exception Code	Exception Message	Description
10000	SUCCESSFULLY PROCESSED	N/N
10325	INVALID ACTION CODE	N/N
10330	INVALID MEMBER DEVICE	N/N
10500	DUPLICATE RECORD	The requested transaction can not be performed, because the data already exists.
10510	RECORD NOT FOUND	The data can not be found
10517	INVALID USER ID ENTRY	User not recognised
10545	PROCESSING TERMINATED - RECORD LOCKED BY ANOTHER USER	The specific record can not be accessed in write or update mode, if it is accessed by another user.
10620	NO MORE RECORDS IN ORDER BOOK	Record not found in Order Book
10809	RESOURCE PRIVILEGE DENIED - MEMBER STATUS IS SUSPENDED	N/N
10870	NO USER SETUP PROFILE ON THIS PARTICIPANT	For the entered user no setup profile is available.
10880	RESOURCE PRIVILEGE DENIED - MEMBER STATUS IS INACTIVE	The specific Resource Privilege is not valid for the entered Member ID, because the member is set to inactive. Inactive members (and their traders) can only perform inquiry functions and change their password.
10881	RESOURCE PRIVILEGE DENIED - MEMBER STATUS IS ACTIVE	The specific Resource Privilege is not valid for the entered Member ID, because the member is set to inactive. Inactive members (and their traders) can only perform inquiry functions and change their password.
10882	RESOURCE PRIVILEGE DENIED - MEMBER STATUS IS INACTIVE	N/N
10890	RESOURCE PRIVILEGE	N/N

Exception Code	Exception Message	Description
	DENIED - MEMBER STATUS IS SUSPENDED	
10891	RESOURCE PRIVILEGE DENIED TO HOUSE	Members of Xetra® are not allowed to trade.
10892	RESOURCE PRIVILEGE DENIED TO THE HOUSE ON BEHALF OF THE MEMBER	The entered Resource Privilege is denied on behalf of the member. Only the functions enter order, delete order and delete quote can be performed on behalf of a member.
10910	RESOURCE NOT AVAILABLE DURING CURRENT TRADING SESSION	The entered Resource is not available depending on the current trading session.
10912	RESOURCE NOT AVAILABLE DURING CURRENT PRODUCT STATE	The entered Resource is not available depending on the current Instrument state, e.g. if the instrument is currently locked by another process.
11001	DATE CANNOT BE GREATER THAN TODAY'S DATE PLUS 1 YEAR	The date entered for a Good Till Cancelled order cannot be greater than 1 year from the current business date.
11002	DATE MUST BE GREATER THAN OR EQUAL TO TODAY'S DATE	An order cannot be entered with a date older than the current business date.
11006	UNAUTHORISED USE OF AGENT ACCOUNT	The user is not set up to trade using account type entered.
11007	UNAUTHORISED USE OF PROPRIETARY ACCOUNT	The user is not set up to trade using account type entered.
11008	UNAUTHORISED USE OF BETREUER ACCOUNT	The user is not set up to trade using account type entered.
11718	INSUFFICIENT RESOURCE ACCESS LEVEL	User has no rights to inquire or modify the selected.
11719	SUBMITTOR IS NOT A SENIOR TRADER OF THE OWNER	User is not a senior trader.
11720	INSUFFIFICIENT ACCOUNT PRIVILEGE	Not enough rights for transaction
11721	MEMBER CHANGES	Member ID is incorrect

Exception Code	Exception Message	Description
	RESOURCE AND TRIES TO TRADE ON BEHALF	
11790	UNKNOWN RESOURCE	Resource not recognised
11795	INVALID MEMBER IDENTIFICATION	Member not known.
12055	ACCESS DENIED	Resource not permitted during current processing
12060	INVALID DATE WAS ENTERED	Incorrect date
12120	INVALID BUY/SELL INDICATOR WAS ENTERED	Incorrect buy / sell indicator
12160	INVALID PRICE WAS ENTERED	Price format not valid
12185	INVALID EXPIRATION DATE	Date invalid for requested transaction
12230	ENTERED QUANTITY VIOLATES LIMIT	Quantity entered exceeds predefined limits.
12232	BID PRICE FAILED REASONABILITY CHECK	This is an informational error to the user warning that the entered limit exceeds a predefined range.
12233	ASK PRICE FAILED REASONABILITY CHECK	This is an informational error to the user warning that the entered limit is below a predefined range.
12234	PRICE FAILED REASONABILITY CHECK	see above
12271	INFORMATION IS RESTRICTED TO SENIOR TRADERS	Request requires senior trader privilige.
12301	BAD MODULE NAME USED FOR CALL - CALL CANNOT BE EXECUTED	not known by order
12417	ENTERED BID QUANTITY VIOLATES LIMIT	Entered bid quantity is greater than the users maximum allowable quantity
12418	ENTERED ASK QUANTITY VIOLATES LIMIT	Entered ask quantity is greater than the users maximum allowable quantity
12419	ENTERED QUANTITIES	N/N

Exception Code	Exception Message	Description
	VIOLATE LIMIT	
12700	INVALID PROCESSING TYPE ENCOUNTERED IN LINKAGE	N/N
12814	ORDER DATA HAS CHANGED SINCE RETRIEVED FOR EDITING	not new for Xetra®
12815	RECORD REQUESTED NOT FOUND ON DATABASE	Record not found
12816	ORDER REQUESTED NOT FOUND ON DATABASE	Record not found
12900	LICENSE REQUIRED TO TRADE IN THIS INSTRUMENT	The entered member or user has no trading permission for this instrument.
12905	BETREUER LICENSE REQUIRED FOR QUOTE MAINTENANCE	The user does not have a betreuer licence in this instrument and therefore cannot enter quotes.
12907	QUOTE REQUEST DENIED TO BETREUER	A user with a betreuer licence in an instrument cannot enter a quote request in that instrument.
12910	FUNCTION REJECTED DURING THIS TRADING PERIOD	Request cannot be processed during this trading period.
12939	INSTRUMENT NOT ASSIGNED TO USER	User has no trading assignments for this instrument.
12940	RESTRICTED - ONLY ALLOWED WHEN TRADING	N/N
12941	INVALID ACCOUNT TYPE	N/N
12942	INVALID FEE TYPE	N/N
12943	INVALID FEE PACKAGE TYPE	N/N
12949	REPORT TO CHANGE NOT FOUND	Report that user wants to modify is not found.
12950	REPORT RECORD IS CURRENTLY LOCKED	Report is currently used by someone else.

Exception Code	Exception Message	Description
12951	REPORT SELECTION WAS ALREADY MODIFIED	Report is modified (reinquire before changing)
12952	NO HISTORIC REPORTS AVAILABLE FOR THIS DATE	N/N
12953	HISTORIC REPORT IS ALREADY REQUESTED	N/N
12954	REPORT TO DELETE NOT FOUND	N/N
12955	REPORT ALREADY EXISTS	N/N
12959	ONLY HISTORIC REQUESTS FOR DAILY REPORTS	N/N
12960	REPORT NOT AVAILABLE	N/N
12962	USER DOES NOT EXIST IN DATABASE	N/N
12963	NO INSTRUMENTS ASSIGNED	N/N
12964	USER ALREADY EXISTS IN DATABASE	N/N
12965	INSTRUMENT ASSIGNMENT ALREADY EXISTS	N/N
12966	DATA HAS CHANGED SINCE RETRIEVED FOR EDITING	N/N
12968	INSTRUMENT ASSIGNED LOCKED IN DATABASE	N/N
12969	INSTRUMENT NOT ASSIGNED TO MEMBER	N/N
12970	TRADE MODIFICATION REQUEST FOR ANOTHER MEMBER	N/N
12971	NOT CURRENT BUSINESS DAY	N/N
12972	TRADE MODIFICATION REQUEST FOR ANOTHER SUB GROUP	N/N

Exception Code	Exception Message	Description
12973	TRADE MODIFICATION REQUEST FOR ANOTHER PARTICIPANT NUMBER	N/N
12974	MAXIMUM NUMBER OF CHANGES FOR A TRADE REACHED	The maximum number of trade modifications has been reached. Currently, a trade can be modified 996 times.
12975	TRADE MODIFICATION REQUEST FOR AN ADJUSTED TRADE	The trade modification is not valid as the trade has been modified in between.
12976	POSTRN RECORD WAS ALREADY MODIFIED	The entered trade record is not valid as the POSTRN record was modified inbetween
12977	POSTRN RECORD CURRENTLY LOCKED	The POSTRN record can not be accessed, as it is currently locked by another process.
12978	TRADE INQUIRY OF A NON CLEARING MEMBER FOR ANOTHER SUBGROUP	The entered trade filter is not valid as a Non-Clearing Trade Inquiry is not allowed to inquire trades of another subgroup
12979	TRADE INQUIRY OF A NON CLEARING MEMBER FOR ANOTHER PARTICIPANT NUMBER	The entered trade filter is not valid as a Non-Clearing Trade Inquiry is not allowed to inquire trades of another participant.
12980	NO TRADES AVAILABLE FOR THIS DATE	For the entered date, no trades are available in the trade book.
12981	NO CLEARING RELATIONSHIP	The entered MemberID is no Clearing Member.
12982	NO TRADE RECORD TO MODIFY FOUND	The entered trade record for modification can not be found in the trade book.
12983	MEMBER ID REQUIRED	A Member ID needs to be entered.
12984	MEMBER IN CLEARING MEMBER FILTER IS NO CLEARING MEMBER	The entered Member ID is no Clearing Member
12985	NO CLEARING RELATIONSHIP BETWEEN MEMBERS IN FILTER	There is no Clearing Relationship between the members submitted in the filter.

Exception Code	Exception Message	Description
12986	MEMBER IN EXCHANGE MEMBER FILTER DOES NOT EXIST	The entered Member ID is not known by the system
12987	TRADE REVERSAL REQUEST FOR AN ADJUSTED TRADE	The trade which should be deleted, has been modified in the mean time.
12988	NO TRADE RECORD TO REVERSE FOUND	The entered trade record which should be deleted can not be found in the trade book.
12989	TRADE RECORD TO BE REVERSED ALREADY DELETED	The entered trade record was already deleted from the trade book.
12990	INVALID TRADER STATUS	N/N
12991	DELETION OF MEMBER SUPERVISOR DENIED	The member supervisor is not allowed to be deleted (as there has to be one user having the rights to operate the system at the member.
12992	AGENT INDICATOR NOT VALID FOR BROKER	The entered Account Type is not valid for a broker. A broker is only allowed to have „P“ as Account Type.
12993	INVALID PROPRIETARY INDICATOR	Proprietary indicator is neither ‘Y’es nor ‘N’o.
12994	INVALID SENIOR INDICATOR	Senior indicator is neither ‘Y’es nor ‘N’o.
12995	INVALID USER FOR MEMBER	N/N
12996	INVALID BETREUER STATUS	Betreuer status is neither ‘Y’es nor ‘N’o.
12997	INVALID MODIFICATION RECORD ALREADY UPDATED	N/N
12998	INVALID MAXIMUM ORDER QUANTITY	Maximum order quantity contains invalid characters.
12999	INVALID AGENT INDICATOR	Agent indicator is neither ‘Y’es nor ‘N’o.
13000	MEMBER ID IS NOT EXCHANGE	The entered Member ID is not the Exchange ID.

Exception Code	Exception Message	Description
13001	INSTRUMENT DOES NOT EXIST	The entered instrument does not exist in Xetra®.
13010	UPDATE REFUSED - REFRESH WINDOW	The data in the database have already updated. Please refresh window in order to get newest data.
13012	DATE MUST BE A BUSINESS DAY	The entered date has to be a business day.
13018	DATA REQUESTED NOT FOUND ON DATABASE	The entered data can not be found in the database.
13019	DAT A FORMAT NOT VALID	The format of the entered date is not valid.
13028	RANGE FORMAT NOT VALID	The entered range has no valid format.
13057	USER DOES NOT EXIST	N/N
13058	INVALID PASSWORD	N/N
13062	REQUEST ALREADY PROCESSED - WINDOW REFRESH MAY BE REQUIRED	Message may be lost, re-inquire to check whether changes occurred.

9.7 LAN Transport Manager

Exception Code	Exception Message	Description
00401	INVALID WORKSTATION ID IN THE REQUEST MESSAGE	N/N
00402	INVALID HOST ID IN THE LINK REQUEST	WS attempted to connect to the MISS with an invalid host ID
00403	MAXIMUM NUMBER OF WS REACHED	Maximum number of WS connected to the MISS reached.
00404	WS SECOND TIME IN XETRA.INI FILE	The WS was registered multiple times in the Xetra® Configuration File.
00405	WS WAS NOT FOUND IN MISS HASH MEMORY	N/N
00406	WS IS NOT IN CONFIG FILE ON THE WS	The WS Xetra® Front End could not start up because the WS was not registered in the Xetra® Configuration File on the MISS.
00407	THE LINK IS ALREADY ESTABLISHED	The WS is already connected to the MISS.
00408	THE WS ID IS TOO LARGE	The WS number within the Xetra® Configuration File was too large.
00422	RUN MISS LAN TRANSPORT MANAGER ONLY ON A MISS	The MISS LAN Transport Manager was started on a WS.
00423	RUN WS LAN TRANSPORT MANAGER ONLY ON A WS	The WS LAN Transport Manager was started on a MISS.
00424	THERE IS NO MISS HOST NAME ENTRY IN THE CONFIG FILE	N/N
00425	THERE IS NO WS HOST NAME ENTRY IN THE CONFIG FILE	N/N
00426	THERE IS NO HOST ID ENTRY IN THE CONFIG FILE	N/N
00427	THERE IS NO WS NUMBER ENTRY IN THE CONFIG FILE	N/N
00431	SERVICE NOT AVAILABLE	Request received for service which is currently not available.

Exception Code	Exception Message	Description
00432	INVALID MISS HOST NAME IN CONFIG FILE	N/N
00433	INVALID MISS ID	N/N
00434	INVALID STATE OF SERVICE	Service is neither available nor unavailable.
00435	INVALID BODY FOR SERVICE NOTIFICATION	Control message has wrong size.
00436	CHANNEL WAS NOT ESTABLISHED	Channel between MISS and WS.
00437	INVALID INTERNAL SLOT NUMBER	Matching request not found for response received.
00438	MESSAGE WITH WS ID NOT DEFINED IN CONFIG FILE	N/N
00439	MISS NAME MISSING IN THE CONFIG FILE	N/N
00440	MISS ID MISSING IN THE CONFIG FILE	N/N
00441	WS NAME IS MISSING IN THE CONFIG FILE	N/N
00442	WS ID MISSING IN THE CONFIG FILE	N/N
00443	SERVICE MISSING IN CONFIG FILE	Not all Xetra® services are in Xervice section of the Xetra® Config File.

9.8 Broadcast Republisher and Receiver

Exception Code	Exception Message	Description
00600	SEQUENCE NUMBER OVERFLOW	Republisher has reached limit of the sequence numbers that it can generate.
00601	THE BROADCAST MESSAGE IS NOT ANY LONGER STORED	Republisher not able to respond on retransmission request.
00602	TOO MANY PENDING BROADCASTS - WILL SHUT DOWN	Buffers of Receiver are full.
00603	OUT OF SEQUENCE BROADCAST RECEIVED	N/N
00604	TOO MANY SUBSCRIPTIONS TO BROADCAST RECEIVER	N/N
00605	REREQUESTED BROADCAST COULD NOT BE FOUND - WILL SHUT DOWN	Receiver cannot obtain retransmission request from republisher.
00606	BROADCAST IS A DUPLICATE OF LAST RECEIVED - IGNORED	A duplicated broadcast message was received and ignored.
00607	MESSAGE RECEIVED WITH UNKNOWN MISS ID	MISS ID not in Xetra® Configuration File.
00608	MESSAGE WAS LOST BETWEEN REPUBLISHER AND RECEIVER	N/N

9.9 VALUES API

Exception Code	Exception Message	Description
00096	XERVICE ID NOT KNOWN	N/N
00119	SUCCESSFUL XETRA STARTUP	N/N
00120	XETRA SYSTEM NOT AVAILABLE ANYMORE	N/N
00121	THE PASSED CALLBACKFUNCTION IS INVALID	N/N
00122	REQUEST MAY NOT HAVE BEEN PROCESSED	N/N
00200	ENVIRONMENT NUMBERS OF ARCHITECTURE AND APPLICATION DO NOT MATCH	The environment number that was passed by the application at connect did not match the environment number of the Xetra Front End.
00201	ENVIRONMENT VALUE IS NOT VALID	N/N
00210	SEND FAILED - ARCHITECTURE MAY BE DOWN	N/N
00211	RECEIVE FAILED - ARCHITECTURE MAY BE DOWN	N/N
00212	SEND TIMEOUT	TCP/IP queue overflow while sending a message to Xetra® Front End.
00213	RECEIVE TIMEOUT	The response to a synchronous request timed out.
00214	CANNOT CREATE COMMUNICATION SOCKET	N/N
00215	ARCHITECTURE IS DOWN - NO CONNECTION TO PEER SOCKET	N/N
00220	APPLICATION ALREADY CONNECTED TO XETRA	The application tried to connect to the Xetra® Front End a second time.
00221	NOT LOGGED ON THE	Submit sent to IBIS-R or to XETRA®

Exception Code	Exception Message	Description
	SYSTEM	without being logged in to the respective *Deutsche Börse Application*
00222	APPLICATION NOT CONNECTED TO XETRA	N/N
00223	CAN ONLY LOG ON ONCE TO SAME BACKEND	The user tried to login to IBIS-R or XETRA® a second time.
00224	CANNOT UNSUBSCRIBE WHEN NOT SUBSCRIBED	The application unsubscribed from an instrument, but did not subscribe beforehand at all.
00225	NOT SUBSCRIBED WITH THIS SUBSID	The application unsubscribed from an instrument without subscribing to that instrument beforehand.
00226	APPLICATION IS STILL LOGGED ON	The application tried to disconnect from the Xetra® Front End but the user was still logged on to IBIS-R or XETRA®.
00227	APPLICATION IS NO LONGER LOGGED ON	N/N
00230	TOO MANY ASYNCHRONOUS MESSAGES PENDING	The maximum number of outstanding responses to application requests has been exceeded.
00231	CANNOT MAP MESSAGE IN TABLE - SO MESSAGE DISCARDED	The received response could not be matched to a pending application request. The response got discarded.
00232	SEQUENCE NO OF SYNC RESPONSE DOES NOT MATCH - SO MESSAGE DISCARDED	An outdated response to a synchronous request was received and got discarded.
00233	PENDING REQUESTS DELETED	The user logged out from IBIS-R and/or XETRA® or disconnected from the Xetra® Front End. Not all responses to pending application requests had been received yet by that time.
00825	“OWN WINDOWS” CONTENT MAY BE OUTDATED	Windows content may be outdated due to gaps in private broadcast.
00826	“MARKET WINDOWS” CONTENT MAY BE OUTDATED	Windows content may be outdated due to gaps in public broadcast.